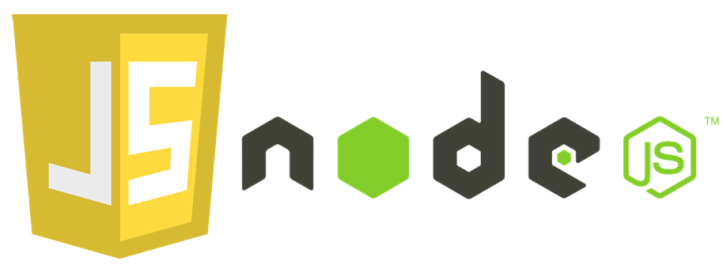
**Period-1 Vanilla JavaScript, Es-next, Node.js, Babel + Webpack and TypeScript-1**

Note: This description is too big for a single exam-question. It will be divided up into several smaller questions for the exam

Explain and Reflect:

* **Explain the differences between Java and JavaScript + node. Topics you could include:**
  + that Java is a compiled language and JavaScript a scripted language
  + Java is both a language and a platform
  + General differences in language features.
  + Blocking vs. non-blocking

**Java** is a compiled language, which means that it is compiled before running. This makes the compiler able to detect errors before runtime.

It’s both a programming language and a platform.

It’s an object-oriented programming language.

It has a virtual machine platform that allows you to create compiled programs that run on nearly every platform.

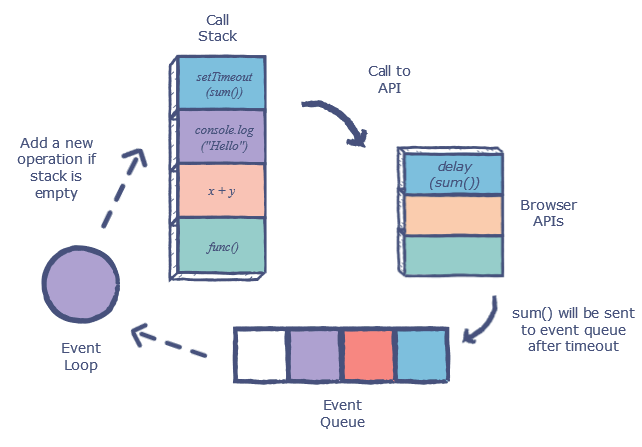
**JavaScript** is a lightweight programming language and used to make web pages interactive. It can insert dynamic text into HTML. JavaScript is also known as a browser's language. JS is not similar or related to Java. They are both used in client-side and server-side Web applications, but there are few similarities only.



* **Explain generally about node.js, when it “makes sense” and *npm*, and how it “fits” into the node echo system.**

Node.js is a js runtime built on chrome's v8-engine. it's used for executing js outside of a web browser

* **Explain about the Event Loop in JavaScript, including terms like; blocking, non-blocking, event loop, callback queue and "other" API's. Make sure to include why this is relevant for us as developers.**



* **What does it mean if a method in nodes API's ends with xxxxxxSync?**

it means that the method will run synchronously instead of async.

* **Explain the terms JavaScript Engine (name at least one) and JavaScript Runtime Environment (name at least two)**

**JS engine:** a computer program that runs js code. All modern browsers have engines to run js (example: chrome's v8 engine), and node.js does the same locally on your pc.

**javascript runtime env:** where our code is executed: chrome, edge, firefox etc. our env has a js engine, as mentioned above.

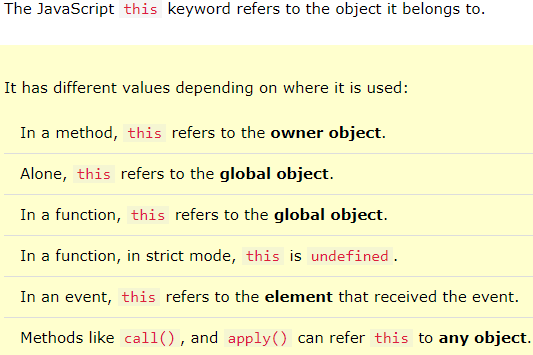
* **Explain (some) of the purposes with the tools *Babel* and *WebPack and how they differ from each other*. Use examples from the exercises.**

**Babel:** Is a Javascript transpiler that converts edge Javascript to ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.

**Webpack:** is the module bundler which will compile all the javascript files or modules into a single file called a bundle.

**Explain using sufficient code examples the following features in JavaScript (and node)**

* **Variable/function-Hoisting**
* Hoisting is a JS mechanism where variables and function declarations are moved to the top of the code-scope before the file is executed. This means that, no matter where the functions and variables are declared, they are moved to the top of their scope, regardless of the scope is global or local
* *this* in JavaScript and how it differs from what we know from Java/.net.



* Function Closures and the JavaScript Module Pattern

**A closure** is a feature in JavaScript where an inner function has access to the outer (enclosing) function’s variables — a scope chain.

The closure has three scope chains:

* it has access to its own scope — variables defined between its curly brackets
* it has access to the outer function’s variables
* it has access to the global variables

**JavaScript Module Pattern** is used to define objects and specify the variables and the functions that can be accessed from outside the scope of the function. We expose certain properties and function as public and can also restrict the scope of properties and functions within the object itself, making them private. This means that those variables cannot be accessed outside the scope of the function. We can achieve data hiding an abstraction using this pattern in JavaScript.

* User-defined Callback Functions (writing functions that take a callback)

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

* Explain the methods map, filter and reduce
* The **map()** method creates a new array populated with the results of calling a provided function on every element in the calling array.
* The **filter()** method creates a new array with all elements that pass the test implemented by the provided function.
* The **reduce()** method executes a reducer function (that you provide) on each element of the array, resulting in single output value.
* Provide examples of user-defined reusable modules implemented in Node.js (learnynode - 6)
* Provide examples and explain the es2015 features: let, arrow functions, this, rest parameters, destructuring objects and arrays, maps/sets etc.
* Provide an example of ES6 inheritance and reflect over the differences between Inheritance in Java and in ES6.
* Explain and demonstrate, how to implement event-based code, how to emit events and how to listen for such events

ES6,7,8,ES-next and TypeScript

* Provide examples with es-next, running in a browser, using Babel and Webpack
* Explain the two strategies for improving JavaScript: Babel and ES6 + ES-Next, versus Typescript. What does it require to use these technologies: In our backend with Node and in (many different) Browsers
* Provide **examples** to demonstrate the benefits of using TypeScript, including, types, interfaces, classes and generics
* Explain how we can get typescript code completion for external imports.
* Explain the ECMAScript Proposal Process for how new features are added to the language (the TC39 Process)

**Callbacks, Promises and async/await**

Explain about (ES-6) promises in JavaScript including, the problems they solve, a quick explanation of the Promise API and:

* ~~Example(s) that demonstrate how to avoid the callback hell (“Pyramid of Doom")~~
* Example(s) that demonstrate how to implement **our own** promise-solutions.
* Example(s) that demonstrate error handling with promises
* Example(s) that demonstrate how to execute asynchronous (promise-based) code in **serial** or **parallel**

Explain about JavaScripts **async/await**, how it relates to promises and reasons to use it compared to the plain promise API.

Provide examples to demonstrate

* Why this often is the preferred way of handling promises
* Error handling with async/await
* Serial or parallel execution with async/await.

Se the exercises for Period-1 to get inspiration for relevant code examples